

Multi-function graphics for a large computer system

by CARL CHRISTENSEN and ELLIOT N. PINSON
Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

I. INTRODUCTION

Although much effort has been spent trying to improve communications across the man-computer boundary, most large computer centers still rely on punched-card-input and line-printer-output for the bulk of their load. Even modern "time-shared" systems rely principally on teletypewriter alphanumeric communication between man and machine.

A great deal has been learned, however, about other ways of communicating with computers, particularly in terms of graphical information.^{1,2,3} On output, graphs and drawings convey meaning to a human viewer much faster than large tables of numbers. Computer generated motion pictures add a time dimension to a display that can help provide physical insight into complex problems.^{4,5} On input, the ability to identify objects in a picture by pointing at them, or to modify a picture in a natural way by drawing in it is a great convenience.^{6,7}

Graphical output has been an important way of presenting results to Bell Laboratories' computer users since the fall of 1961, when a Stromberg Carlson 4020 microfilm printer was installed at Murray Hill. It has allowed users to replace pages of tabular data with pictures that are far easier to understand. Microfilm picture generation was running recently at a rate of about 500,000 frames per year. In addition, computer movies were being generated at a rate of about 1,000,000 frames per year.

In 1964 the experimental GRAPHIC-1 display terminal⁸ was interfaced to a 7094 at Murray Hill to provide semi-on-line graphical input-output to the 7094 for a single user at a time. Despite the fact that this was a one-of-a-kind system, it was extremely popular and heavily used. It incorporated a small processor in the terminal to handle all real time requirements generated by the user.

More recently, an experimental on-line graphical output device that can produce hard-copy at a rate of

about one page every five seconds was interfaced to the 7094. This device uses a microfilm intermediate stage with a rapid high temperature developing process to produce output within 30 seconds of the film being exposed. This, too, has been popular with users, although mechanical problems have caused excessive down-time for the device.

Although the graphics facilities on the 7094 system were extremely useful, they were far from what a modern, large-scale computing center should offer its customers. Among the features that seemed ready for improvement were the hard-copy output turnaround time (frequently overnight on the SC-4020, although sometimes as good as a few hours), the fixed geographical position of all devices, the inaccessibility of the system through standard transmission facilities, and the extremely small number (in fact, just one) of terminals from which users could run their graphical programs.

In Section II of this paper an overview is given of a new graphics system being developed at Bell Laboratories. Following sections describe certain aspects of the system in more detail. Section III describes hardware, and Section IV describes the Graphical Data Structure used to represent picture information in the computer. The GRIN-2 graphical programming language is discussed in Section V (an annotated example of a GRIN-2 program is given in the Appendix). Section VI describes some functions performed by the GRAPHIC-2 Executive System, and Section VII describes briefly some of the functions of the GE-645 Software System. A brief summary is given in Section VIII.

II. System overview

A modern large scale computer facility should provide a variety of graphical services for its customers. These range from hard-copy output of high quality to highly interactive on-line graphical communication be-

tween man and computer. The graphical services provided by the system described in this paper include:

1. Rapid hard-copy output (STARE system);
2. On-line direct-view cathode ray tube output (GLANCE system) as an accessory display for a teletype console;
3. Remote on-line interactive graphical input-output (GRAPHIC-2 system);
4. Precision microfilm and hard-copy output;
5. High precision drafting quality output.

Obviously, no single type of display can provide these very different services in an economical way. Rather, several different kinds of equipment are required. Services 4 and 5, although very important, will not be discussed further since they are provided by standard commercially available equipment, namely, an SC-4060 microfilm printer and a Gerber drafting table. As will be seen below, however, programming support for these devices is handled in large part by software used in common for all hardware systems. The first three categories involve systems for which a combined hardware-software design effort was performed. Hardware details for each of these systems (STARE, GLANCE, and GRAPHIC-2) are given in Section III.

The graphics systems will run on a large multi-access computer installation at Bell Laboratories' Murray Hill facility. This computer system will soon be operating under the Multics⁸⁻¹³ system. The equipment features a dual-processor GE-645 computer with 256K of 36-bit 1 μ s core memory and extensive on-line mass storage (drum, disk, etc.). Multics system features include a large on-line multi-level file system, memory segmentation and paging (to give each user an extremely large virtual memory space), and the ability to communicate with many on-line communication devices. The system will simultaneously service many console users and programs entered in a conventional "over-the-counter" mode. Most on-line consoles will be of the teletypewriter variety. Additional graphic consoles as described in this paper will also be available. Response times for users of the graphics system will be measured in seconds rather than minutes (for GRAPHIC-1) or hours (for the SC-4020). True remote access to the computer via dialed up voice grade phone lines will be possible from several of the terminals.

For computer graphics to have a substantial impact on many people, it is necessary to make graphic equipment available and accessible. Two factors must be considered: first, physical accessibility, made possible by providing multiple devices which can be used simultaneously and which are conveniently located; second, program accessibility made possible by providing the user with a powerful set of system functions and a higher level language for his graphical programming (see Sec-

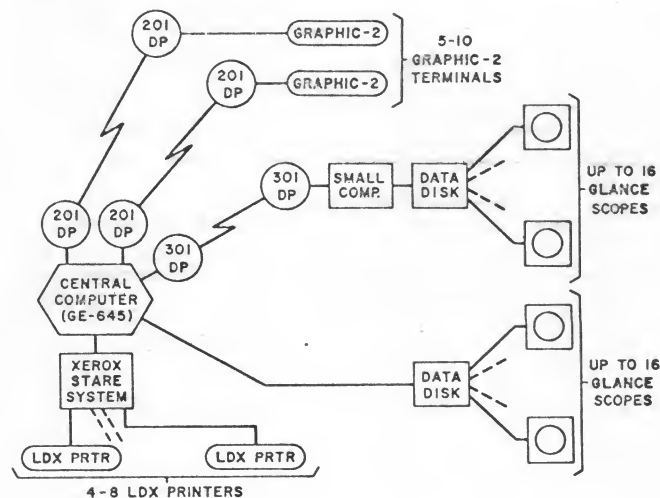


Figure 1—On-line computer graphics facility

tion V on the GRIN-2 language). Although specific numbers of devices are not known at this time, Figure 1 shows a configuration that should be achieved during 1968.

Despite the fact that many different types of hardware are supported, the bulk of the software in the central computer is device independent. This is because a device independent representation of picture information is used in the central machine. This representation, called the Graphical Data Structure (see Section IV) describes a picture that can be displayed on any of the graphical terminals in the system. The Graphic Data Structure provides a standard for communication of picture information between programs and is the form in which pictures are stored in the GE-645 File System. Device dependent Translator modules are used to convert from the central format to the command formats required by a particular device. There is one such Translator module for each type of graphic device in the system.

In the case of the STARE hard-copy system, a user will be able to go to an output station near his office rather than have to go to a single central distribution point. Up to eight such remote printer stations may be served by the hard-copy system. Among other uses, the rapid hard-copy facilities are expected to aid users accessing the machine from teletype consoles whose low speed makes them unsuitable for listing alphanumeric files. Dumps and other listings can be produced on the STARE system upon command from a teletypewriter console.

The intent of the GLANCE on-line direct view CRT system is to provide an accessory graphical display capability for standard teletype consoles. The CRT dis-

play is physically independent of the teletypewriter, although located immediately adjacent to it. Under software control, graphical output from the program being controlled from the teletype will be displayed on the CRT display. The CRT display can be maintained without consuming costly central machine resources, since a special high-speed disk buffer is used to store the commands that generate the picture. A track of this disk has to be loaded once for each picture to be shown. Thereafter, the picture is maintained on the display tube without central machine intervention. Since a high data rate is used between the disk buffer and the displays it supports, each display must be within coaxial cable reach of its buffer. A complete GLANCE system (disk plus CRT terminals) can be located remote from the central computer if a small computer is added to the system. The small computer can receive coded graphical data at low speed over a transmission link, convert it to the command set required by the display logic, and write a picture at a time at high speed onto the magnetic disk. An example of this type of operation is shown in Figure 1.

The GRAPHIC-2 system is an interactive graphic input-output terminal. Graphical input is accomplished by using a light pen to point at objects displayed on a CRT. This system accesses the central computer over a voice grade transmission line. One of the principal objectives of the GRAPHIC-2 system is to provide good terminal response to the operator while using slow (but economical) communication lines to the central computer system. Unfortunately good terminal response is measured in fractions of a second, while response from the central computer may take several seconds. For example, transmission of a 100-word data block (18 bit words) takes almost one second over the 2000-baud voice grade line. Added to this must be the delay involved in capturing the central processor (which is simultaneously trying to service numerous other users) and in carrying out central machine computations.

The solution to this problem lies in having GRAPHIC-2 itself handle users' real time demands. Thus, such actions as pen-tracking and drawing are handled in real time at the display terminal (by the PDP-9 processor that is part of the GRAPHIC-2). It is only the results of these actions (final X-Y coordinates or identity of a selected object) that are sent back to the central computer, where a master copy of the picture is updated. Fortunately, modifications to this master copy do not have to be made in real time since the user at the terminal sees the results of his actions on the local version immediately.

Two versions of a user's program are compiled. These programs originate from a single GRIN-2 (see Section V) source program. Both compilations are performed

by the GE-645. One produces GE-645 machine code and the other produces PDP-9 machine code. The former executes in the GE-645, operates on the *Graphic Data Structure* (see Section IV) in the central machine and obtains its "pseudo real-time" inputs from a queue sent back from GRAPHIC-2. The latter executes in a GRAPHIC-2 terminal, modifies a local (at GRAPHIC-2) copy of the data structure and services real-time inputs as they occur (light-pen strikes, button pushes, keyboard inputs), simultaneously recording their occurrence on the queue destined for the GE-645. The queue is periodically sent to the central computer, allowing the GE-645 execution of the user's program to catch up with GRAPHIC-2. This strategy minimizes the amount of data that have to be transmitted from GRAPHIC-2 to the central machine, since only the real time events are transmitted.

III. Display hardware systems

The following three display systems are described:

- A. The STARE (Short Turn-Around REcorder) rapid hard-copy system.
- B. The GLANCE (Graphical Accessory on-line Console) cathode ray tube output display.
- C. GRAPHIC-2, a graphical input-output display terminal that includes a small processor, display CRT, light-pen and keyboard.

A. The STARE hard-copy system

The STARE hard-copy facility accommodates up to eight rapid hard-copy output stations. The organization of the system is shown in Figure 2. The system consists of multiple remote Xerox LDX (Long Distance Xerography) Printers interfaced to a central buffer memory and control unit that is located near the GE-645. The LDX printers form an image by a raster scan technique,

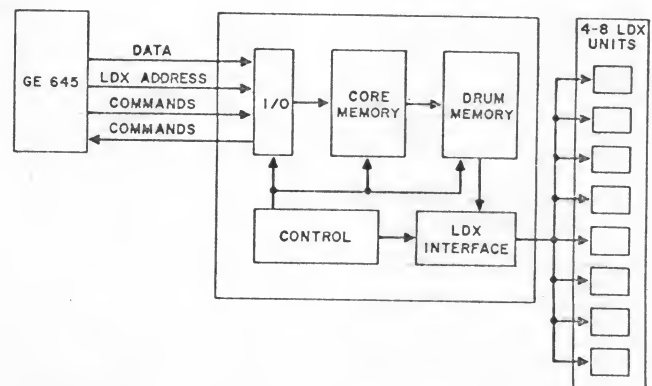


Figure 2—STARE system organization

so the data from the central buffer unit to the printers are essentially a video signal for controlling the intensity along each scan line. These data to the LDX Printer are transmitted from the central buffer at about a 250,000-baud rate over coaxial cable links. There are eight separate buffer areas on the magnetic drum which function independently. Therefore, transmission to up to eight printers can take place simultaneously.

The central buffer unit is a digital scan-converter. It accepts drawing commands in an incremental command code (the same one used by the GLANCE system described below) and converts these to the raster scan required by the LDX Printers. The complete picture is specified on a plotting grid 1024×1408 (corresponding to a plotting area of $8'' \times 11''$). The core memory contains enough storage to represent the raster scan of about a one inch strip of the output (128 lines by 1024 bits/line). An I/O command from the central computer specifies which strip the core buffer is to scan-convert. The incremental commands that specify the entire picture are then sent to the buffer unit. The buffer logic transforms these to raster scan bits for the proper strip and stores them in the core buffer, 1 bits representing black dots and 0 bits representing white. After the picture has been scan-converted, the core image is automatically transferred to the appropriate tracks on the magnetic drum. This process is repeated eleven times to transfer the entire picture to the drum. An I/O Print Command is then issued to the computer which starts the appropriate LDX Printer and initiates the drum-to-printer transfer.

The data rate between the central computer and the core buffer unit is about 2×10^6 baud. Since a complete picture is often specified in under 10^5 bits in the incremental code, each burst of data takes about .05 seconds for the core to core transfer. The buffer-core to drum transfer is at about a 10^6 baud rate, and takes about .15 seconds to complete (including drum latency). A full picture (11 strips), by these rough calculations, takes about 2.2 seconds to transmit from central computer to drum buffer. About an additional five seconds are required for the drum to LDX Printer transfer. Thus, a single printer can produce about 8 frames per minute in this system, and a maximum throughput of over 24 frames per minute distributed among several printer stations is possible.

B. The GLANCE direct-view scope system

The GLANCE system has been described in a previous paper.¹⁴ However, the essential elements will be repeated here for the reader's convenience. Figure 3 shows the GLANCE configuration. A central disk buffer holds picture information for up to 16 GLANCE display terminals. A single track of the disk is dedicated

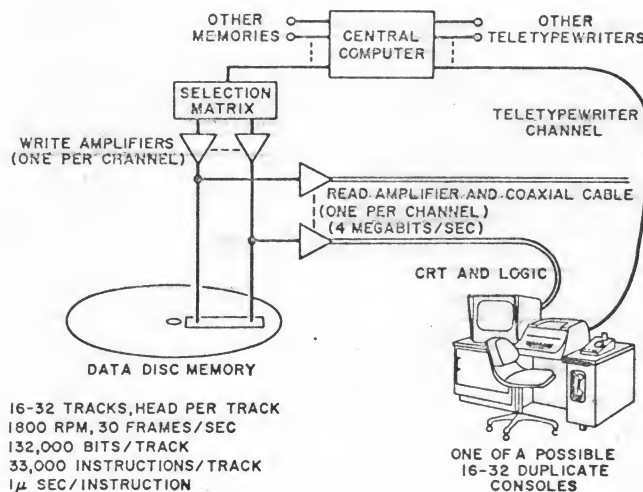
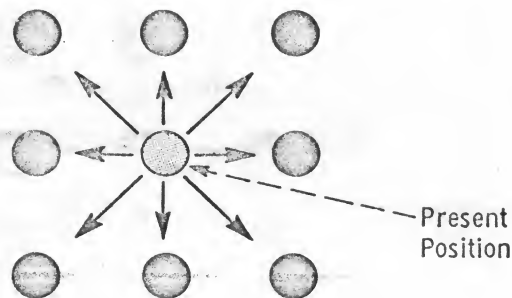


Figure 3—GLANCE system organization

to each of the terminals, and each of the tracks has its own read-head and line driver. The picture on each track is specified in terms of the simple four-bit incremental commands shown in Figure 4. These commands are decoded by logic in each of the scope terminals where X-Y deflection and unblanking signals are produced to position and display points. The commands allow the beam to be moved a certain number of units in any direction on a 1024×1024 grid (left-right, up-down, or diagonally) covering a $10'' \times 10''$ area on



COMMANDS

Control	Moves
No op	Right
Reverse beam on/off	Right-Up
Increment intensity (mod.4)	Up
Set scale x 1	Up-Left
Set scale x 2	Left
Set scale x 4	Left-Down
Set scale x 8	Down
Set scale x 16	Down-Right

Figure 4—Incremental command plotting technique

the scope-face. Moves can be made with the beam blanked to reposition it without leaving a trace. Four spot intensities can be used. A scale-factor can be preset so that incremental commands cause a beam move of one, two, four, eight, or sixteen grid units. This permits fast slewing with the beam blanked.

The disk rotates at 30 revolutions per second (1800 rpm), so the picture on each scope is refreshed at a 30-cycle rate. With the proper phosphor this rate is high enough to give a flicker-free display. The data transfer rate from disk to display scope is 4×10^6 baud, so this connection is made with coaxial cable. The display terminals can be located up to a mile or so cable length from the central disk buffer. At the data rate used, 10^6 points/second are plotted by the display terminal. Figure 5 shows a picture of a GLANCE console including a Teletype, and Figure 6 shows pictures taken directly from a GLANCE CRT.

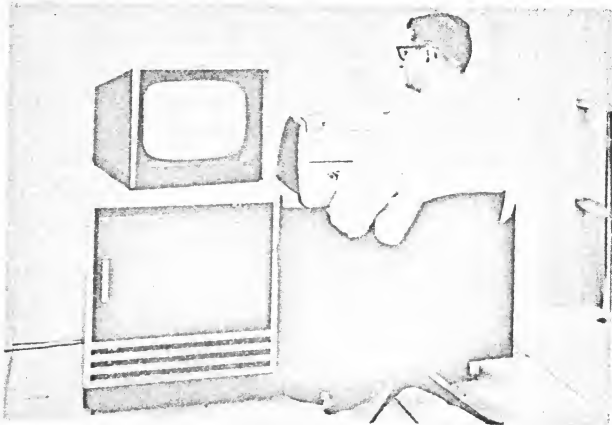


Figure 5—The GLANCE terminal

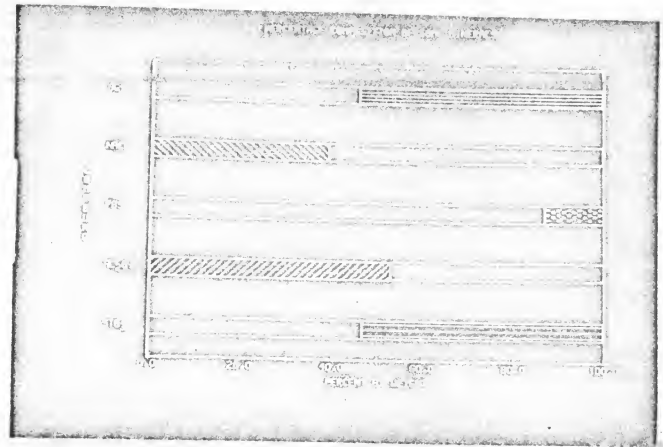
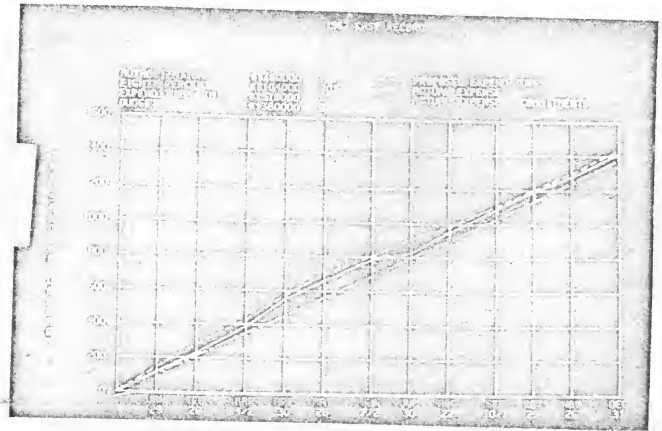


Figure 6—Sample pictures from a GLANCE CRT

C. GRAPHIC-2

Many of the factors that affected the design of GRAPHIC-2 were discussed in Section II of this paper. A detailed description of the system is presented in a forthcoming paper.¹⁵ Only the essential elements will be repeated here. Figure 7 shows the GRAPHIC-2 organization. A picture of the hardware is shown in Figure 8. A standard DEC PDP-9 with optional automatic priority interrupt and a direct memory access channel is the heart of the system. Its memory consists of 8K words of 18-bit one microsecond core. This memory serves as display buffer memory and also holds programs that execute in the PDP-9. An optical paper tape reader and a paper tape punch are standard. The display scope and display command set are described in detail in the forthcoming paper.¹⁵ A light-pen and keyboard (ASCII alphanumeric plus 8 pushbuttons) complete the GRAPHIC-2 hardware.

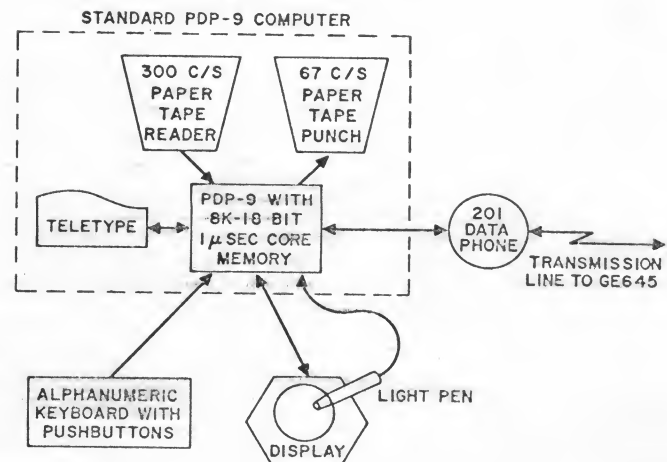


Figure 7—GRAPHIC-2 organization

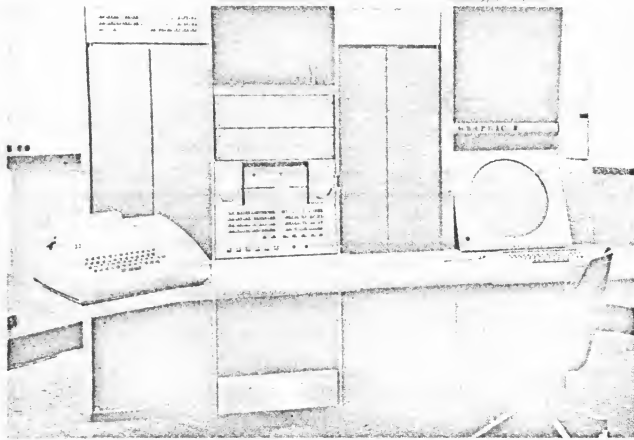


Figure 8—The GRAPHIC-2 terminal

The scope has a character generator that can display the full ASCII 96 graphic set. Among the features that have been included in the scope hardware to facilitate programming are: Edge violation traps whenever a command tries to make a line cross a scope boundary, either in or out; software commands to determine the exact state of the scope at any time and to restore the scope to any state; commands which when encountered automatically stop the scope and interrupt the PDP-9 processor. In short, a great deal of thought was devoted to the design of a scope-processor system that is easily programmable.

IV. The graphic data structure

The various graphical devices that are supported by GRAFCOM (GE 645 Graphical Communication software) function quite differently at the hardware level. In particular, they use different command formats to generate pictures. However, in the GE-645 a single standard way of describing graphical information is used that is independent of any particular display device. This common representation is called the Graphic Data Structure. All picture building routines generate blocks that are attached to the user's graphic data structure. All picture editing routines assume they are operating on pictures represented in this standard way. Thus a single set of graphical programs is used to manipulate graphical information whether a GRAPHIC-2 interactive console, or any other graphic output device is being used. Before display commands are sent to a particular type of device, a device dependent program is called to convert from the standard format to the special command set used by the device. Methods for representing picture structure have been described in the literature.^{1,16,17} The structure used in GRAFCOM differs from these in detail, but was strongly influenced by several previous systems.

Although the main graphical data base is in the large central computer, the GRAPHIC-2 terminal stores picture information in an identically structured form. As will be seen, information in the graphic data structure is stored in discrete blocks that are linked together by chains of pointers. Although formats within blocks in the GE-645 are different from those in the GRAPHIC-2, there is a one-to-one correspondence between blocks in the two structures. Because the memory in the GRAPHIC-2 is small (8K, 18-bit words), all blocks in the data structure do not have to be present in GRAPHIC-2 at the same time. A dynamic memory management system fetches blocks from the central computer when they are needed, and releases memory in GRAPHIC-2 to free storage when blocks are no longer needed. Some differences between block formats in the central machine and in the GRAPHIC-2 are explained at the end of this section. The following discussion describes the graphical data structure in the central computer.

Blocks in the structure

The graphic data structure contains four types of blocks: *node blocks*, *branch blocks*, *leaf blocks* and *data blocks*. These are shown symbolically in Figure 9. (Although a branch is represented by an arrow, it is not simply a pointer, but is a contiguous set of memory locations, as are each of the other block types.) Nodes

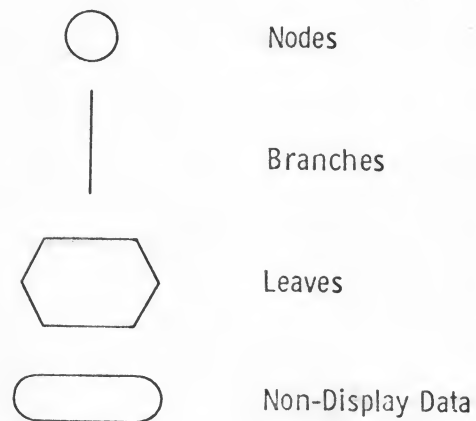


Figure 9—Blocks in the graphic data structure

and branches are blocks of fixed size; other blocks may be of arbitrary length. The generic form of graphical structures is the *directed graph* (with no closed loops). A directed graph is a set of *nodes* and connecting *branches* in which the branches have a direction as-

sociated with them, i.e., they point from one node to another. Some of the terminal nodes of the directed graph have special significance and are called *leaves*, in analogy with the way leaves are extremities on a tree. By convention, only leaves in the structure can specify displayable material. Other blocks are present only to provide structural information. Figure 10 shows the typical form of a graphical data structure.

Each node represents a particular sub-part of the picture. Each branch represents a particular occurrence, or *instance*, of the node to which it points. The branch may be thought of as a graphical call to a sub-part of a picture. It will cause the sub-part to appear at a position specified by information in the branch. Nodes serve to group together those sub-parts that the user may want to be associated with each other to form a larger sub-part. This nesting can continue to arbitrary depth.

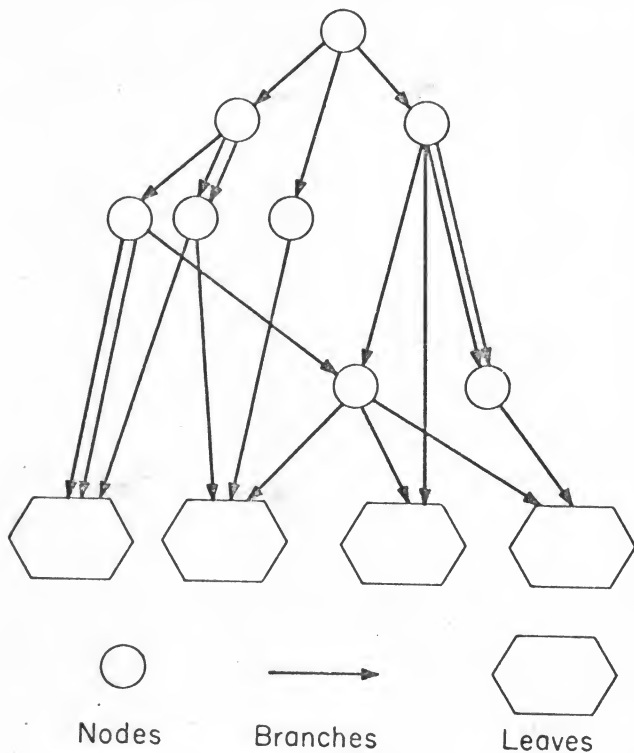
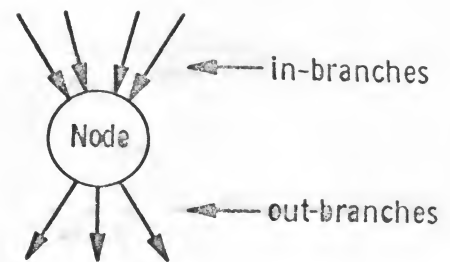


Figure 10—Typical graphic data structure

Nodes

Figure 11a shows the configuration of a typical node. Note that an arbitrary number of branches may enter a node, and an arbitrary number of branches may leave. It would not be possible to allocate a fixed size block for the node if pointers to all these branches were required in the node. To avoid this problem, the "in-branches" (branches entering the node) and "out-branches" (branches leaving the node) are organized



NODE BLOCK CONTENTS

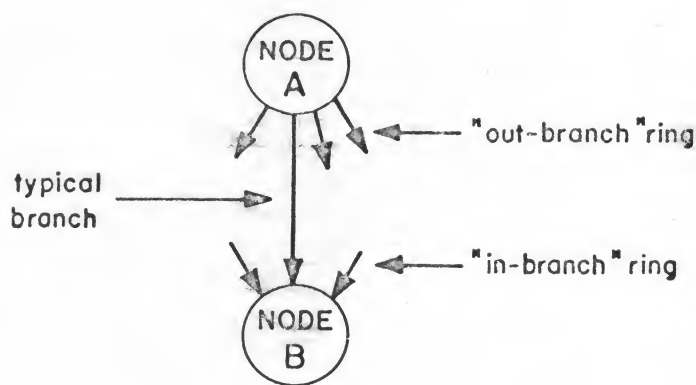
- Block type identifier
- Pointer to name data-block
- Pointer to first "in-branch"
- Pointer to last "in-branch"
- Pointer to first "out-branch"
- Pointer to last "out-branch"

Figure 11—Typical node-block and its contents

in separate "rings."¹⁷ The node contains a pointer to the first branch in each of these rings, and each of these branches contains a pointer to the next branch, etc. Finally, the last branch contains a pointer back to the original node. A thread established by "back-pointers" also links all the elements in a ring in order to make tracing through rings faster for computer programs. Figure 11b shows the contents of a node block. In addition to pointers defining the "in" and "out" branch rings, each node contains a block type identifier (identifying it as a node) and a pointer to a data block that contains the symbolic name of the node. Thus every node (and therefore every picture subpart) may be referred to symbolically if desired. If the node has not been assigned a name, this pointer will contain a null value.

Branches

Figure 12a shows a typical branch connecting two nodes. Each branch has two rings passing through it: the "out-branch" ring associated with the node the branch starts on, and the "in-branch" ring associated with the node the branch terminates on. Figure 12b



BRANCH BLOCK CONTENTS

- * Block type identifier
- * Name-data-block pointer
- * "In-branch" ring pointers
- * "Out-branch" ring pointers
- * System non-display-data pointer
- * User non-display-data pointer
- * Branch x-displacement
- * Branch y-displacement
- * Branch display control parameters

Figure 12—Typical branch-block and its contents

lists the contents of a branch block. It contains a block type identifier (identifying it as a branch) and a pointer to a data block that contains its symbolic name. Thus every branch (and therefore every instance in a picture) may be referred to symbolically. If the branch has not been assigned a name, this pointer contains a null value. Next, the branch contains two ring elements. Each ring element contains three pointers: (1) a pointer in a forward chain linking all branches in the ring; (2) a pointer in a backward chain linking all branches in the ring; (3) a pointer to the node that is the "head" of this ring. These pointers facilitate rapid examination of the structure under program control.

Two pointers are included to data-blocks. These are the "system non-display data" pointer and the "user non-display data" pointer. If either of these pointers is not being used, a null pointer value is used for it. The system pointer and any data blocks linked to it are for the exclusive use of the graphical programming system. User programs may not access this information. The principal use of the system pointer is to point to a data block that identifies the branch as a light button. That data block contains the name of a program entry

point that control is to be passed to if the instance represented by the branch is pointed at by a light-pen.

The purpose of the user non-display pointer is to allow the user (programmer) to attach non-display information to the graphic data structure. This non-display information is placed in "data-blocks," which are one of the four types of blocks found in the graphical data structure. These data-blocks are for the exclusive use of the programmer. They are defined and allocated under program control and may be of arbitrary format and size. They are intended to allow the user to specify information about instances that does not appear in the display.

Finally, the branch contains three fields necessary for the generation of the display itself. The X and Y displacements specify the distance between the display origins of the nodes (or node and leaf) which the branch joins. Since all display information in the structure is in terms of relative coordinate information (only the starting point of a display is an absolute coordinate), instances of sub-pictures can be moved about the display surface simply by changing the X and Y displacements in the proper branch. The last field contains display parameter information. Such things as intensity, scale and whether or not the light-pen is to be enabled during the display of the instance can be specified.

Leaves

Leaves specify how portions of a picture are actually to be drawn. They contain the "ink" that is visible in a portion of a picture. Data is placed in leaves by calls to data generating subroutines in GRAFCOM. Both text and line drawing information may be specified. When information is placed in leaves, the programmer is specifying information on a very large display surface. It is only when the information is finally displayed on one of the several available devices that a "window" is specified that establishes what portion of the display is to be visible on the output device. Data "grown" into leaves in real time by drawing on the GRAPHIC-2 scope must be rescaled before it is stored in the central data structure. Since a leaf may contain an arbitrary amount of data, its size cannot be specified in advance. For this reason, the leaf is broken into blocks of two types; the leaf-header block and leaf-data blocks. The leaf-header block contains a single ring-element, the "in-branch" ring, since by convention no branches point away from a leaf. It also contains a pointer to the first leaf-data block and to the last leaf-data block that make up the body of the leaf. These leaf-data blocks are linked by chain of pointers. A new leaf-data block is allocated and added to the chain whenever new data for a leaf are generated.

Example

The following example illustrates many of the features of the graphic data structure. Figure 13a shows a type of pattern familiar to anyone who has studied electronics. It is a diagram of a simple resistor-capacitor circuit. Because of generally accepted conventions, the meaning of this picture is immediately apparent to many people. Computers, however, have taken no electrical engineering courses. The pattern of lines might just as well be abstract art so far as the computer is concerned. The fact that a certain group of straight line segments symbolizes a resistor is not inherent in the picture, but is an organization imposed on the image by the viewer.

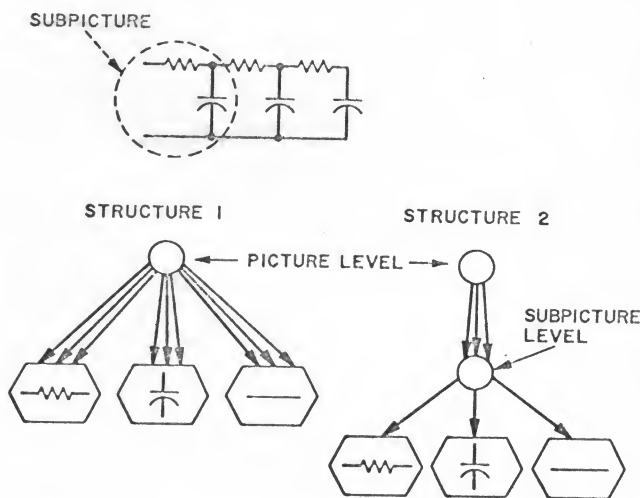


Figure 13—Simple circuit and two possible structures for it

A leaf is generated that contains the picture corresponding to each of the three types of elements in the circuit: resistors, capacitors and short-circuits. These elements may be positioned anywhere on the display surface by appropriate connecting branches. In the representation of Structure 1, shown in Figure 13b, there is no additional structure in the picture aside from the fact that each element type is a single leaf. In the representation of Structure 2 in Figure 13c, a sub-picture consisting of a resistor, capacitor and short circuit has been formed. In the whole picture there are three instances of this sub-picture (one for each of the branches pointing to the subpicture node), and one instance of each of the three basic elements.

Additional non-display data blocks would undoubtedly be linked to these branches in most real applications. They would contain such information as the electrical type of the elements being displayed and their values for use by analysis programs.

Modifications for GRAPHIC-2

The data structure in GRAPHIC-2 retains all the structural information present in the data structure in the central computer. GRAPHIC-2 is the only one of the display systems that uses information in structured form. All the others simply receive a set of display commands from the appropriate device Translator module. GRAPHIC-2 needs the structure because of the real time response it must give users at its console. It must be able to identify objects that are pointed at without requiring the intervention of the central computer. It must be able to edit the structure, too. Thus, the GRAPHIC-2 data structure contains nodes, branches, leaves and data blocks that are in one-to-one correspondence with equivalent blocks in the central data base. Formats internal to these blocks are different, however. First of all, leaves in GRAPHIC-2 contain display commands in the format required to run the display scope, while picture information in the central computer is represented in a device independent way. Because a different dynamic storage allocation technique is used in GRAPHIC-2, a leaf is a single contiguous block of memory rather than a sequence of leaf-data blocks linked to a leaf-header. These leaves can be grown in real time under program control and may be of arbitrary length. Because space is a scarce resource in the small GRAPHIC-2 computer, an abbreviated pointer system is used to link blocks in the data structure. In particular, back pointers and pointers to the heads of rings are not used. Tracing one's way through the structure therefore may require more time, but time is a resource that is more readily available than space in GRAPHIC-2.

V. GRIN-2 language

The GRIN-2 (*GR*aphical *IN*teraction) language is a high-level graphical programming language that permits the generation and manipulation of the graphical data structure, and provides statements for controlling real-time man-machine interaction. The interaction portion of the language is used with the GRAPHIC-2 graphical terminal. The rest of the language pertains to the common data structure used by all graphical devices and terminals.

The following is not a specification of the GRIN-2 language, but a brief description of a few statements in several categories, the purpose of which is to give the reader the flavor of the language and allow an understanding of the programming example given in the Appendix. The categories described are:

- (1) Real-Time Man-Machine Interaction
- (2) Structure Generation
- (3) Display Data Generation

- (4) Structure Editing
- (5) Display Control

Real time man-machine interaction

The basic man-machine interaction philosophy of the GRIN-2 language is one of the question and answer. The language provides statements which ask questions of the operator at a GRAPHIC-2 terminal. GRAPHIC-2 then waits (with control held by the language statement) until the operator answers the question by some light-pen or keyboard action. Control is then passed on, along with his answer, to other GRIN-2 statements which use the answer to direct further processing.

One of the real-time questions provided in GRIN-2 is asked by the WHICH statement. WHICH asks the operator the question "which of the objects displayed on the scope?" The operator then answers by pointing the light-pen at one of the objects. The meaning of this question depends on when it is asked. For instance, if a program to delete objects from the scope is in control it might use the WHICH statement to ask "WHICH of the objects displayed on the scope should be deleted?" When the operator pointed to one of the objects with the light-pen, control would pass on from the WHICH statement to the next statement. The next statements would then use the answer to delete the object picked.

The WHICH statement can also be used to direct the flow of control in the program through the use of light buttons. A light button is a displayable object which is associated with a program or subroutine. Given a number of light buttons displayed on the scope, the program in control could use the WHICH statement to ask "WHICH program should receive control next?" or "WHICH function should be performed next?" The operator could then pick the desired light button with the light-pen, and the WHICH subroutine would pass control to the picked light button's associated program.

Another basic real-time question in the GRIN-2 language is asked by the WHERE statement. WHERE asks the question "where on the scope?" by displaying a tracking cross which can be moved around the scope face with the light-pen. The operator answers this question by positioning the tracking cross with the light-pen. When the tracking cross is positioned where he wants it, the operator informs GRAPHIC-2 by pushing a button or taking some other action that can be specified as an argument in the WHERE statement. This action could even be picking a light button with the light-pen. Objects displayed on the scope can be made to move with the light-pen by "attaching" them to the tracking cross. Other GRIN-2 statements use WHICH and WHERE to form higher level statements such as DRAW. DRAW uses repeated calls to WHERE

to allow the operator to draw a broken line segment on the scope.

Structure generation

Building a graphical data structure is done with three statements: LEAF, BRANCH, and NODE.

LEAF BLKPTR

generates an empty leaf-block and places a pointer to it in BLKPTR.

BRANCH (BLKPTR, FROM, DXY, PARM, TO, LB, DATA)

generates a branch-block and places a pointer to it in BLKPTR. FROM is a pointer to the node from which this branch is to descend. TO is a pointer to the leaf or node to which this branch is to point. DXY is a pointer to a DX, DY word pair which specifies the relative displacement on the scope between the display origins of the FROM and TO nodes. PARM is a pointer to a location which contains the parameters to be specified in the branch. If LB is given, it specifies that the branch is a light button and its associated subroutine is LB. DATA points to a user's data block that is to be attached to his branch. All of the arguments in the BRANCH statement are optional.

NODE BLKPTR ((B.DXY, P, T, LB, D) (. . .) . . .) generates a node-block and places a pointer to it in BLKPTR. The next string of arguments specify a branch or set of branches which are generated and descend from the generated node. These arguments are similar to those in the BRANCH statement.

Display data generation

Display data in leaves are the scope commands which drive the scope and produce the picture. The following statements produce display data which is directed into the leaf specified by the last occurrence of either an INLEAF or LEAF statement (last generated leaf).

TEXT (CH1, CH2, CH3 . . .)

stores a series of characters in the last specified leaf. When the leaf is displayed, characters CH1, CH2, CH3 will appear on the scope.

VECTOR (DXY1, DXY2, DXY3 . . .)

generates a series of vectors. The arguments may be pairs of signed decimal integers or the location of a DX, DY word pair.

Structure editing

Statements in this category allow changes to be made to an already existing structure.

TCMOVE (BRANCH1, BRANCH2, . . .)

This is a dynamic statement which specifies that DX and DY (relative displacement) in the branches BRANCH1, BRANCH2 etc., are to be changed as the position of the tracking cross is changed in the next

WHERE statement. This allows objects to move with the tracking cross.

DETACH (BRANCH1,BRANCH2, . . .)

Specifies that the branches BRANCH1, BRANCH2, etc., should be detached from the structure.

ATTACH BRANCH,NODE

Attaches the branch specified by the pointer BRANCH to the node specified by the pointer NODE.

Display control

The structure or structures to be displayed are specified by attaching them to a special node called the DISPLAY NODE (DSPNOD). This may be done with the standard structure editing statements or by the special display control statements described below.

NEWDSP (BRANCH1,BRANCH2, . . .)

Any existing branches which are attached to the display node are detached and BRANCH1,BRANCH2, etc., are attached to the display node.

ADD DSP (BRANCH1,BRANCH2, . . .)

The branches BRANCH1,BRANCH2, etc., are attached to the display node without disturbing any branches which were already attached to it, adding them to the picture.

SUB DSP (BRANCH1,BRANCH2, . . .)

The branches BRANCH1,BRANCH2, etc., are removed from the display node without disturbing other attached branches (subtracting them from the picture).

Other categories of GRIN-2 statements include STRUCTURE TRACING, NON-DISPLAY STRUCTURE GENERATION, and SYSTEM CONTROL.

VI. Graphic-2 executive

The GRAPHIC-2 Executive is a set of programs that execute in the GRAPHIC-2's PDP-9 processor. It performs the functions of memory management, interrupt handling, data communication and display management. These functions are described below.

Interrupt handling

The interrupt handler is the heart of the executive program. All of the operator inputs to the GRAPHIC-2 (light-pen, pushbuttons, keyboard) are detected via interrupts as are conditions such as dataphone input and output, oscilloscope edge violation, and the occurrence of scope commands with an interrupt bit set. When an interrupt occurs, the trap handler gives control to the subroutine associated with that interrupt. This association is changeable from outside the trap handler, allowing programs to specify and change the reaction to any of the interrupts.

Data transmission

A 201B Dataphone (2,000 bits/sec) is used to transmit data back and forth between the GRAPHIC-2

and the GE-645. The data communication program controls the dataphone, buffers input and output data streams, detects errors and provides an error recovery strategy.

Memory management

The memory management programs give the PDP-9 a very large virtual memory, using the GE-645 and its file system for secondary storage. To make this possible the user's program, data and graphical data are broken up into blocks of variable size. Each block is given a unique 17-bit identification number (ID) before it is transmitted to the GRAPHIC-2, and all references between blocks are in terms of these ID's and an offset within the block. The ID is also used to request a block from the GE-645. To make interblock referencing practical, every block in the GRAPHIC-2 memory has assigned to it while and only while it is in core an entry in the BLOCK TABLE, which is located in the GRAPHIC-2 (see Figure 14). This entry establishes the correspondence between a block's ID and its location in the GRAPHIC-2, which makes possible easy relocation of blocks in core; only the core address in a block's entry in the block table need be updated when a block is moved.

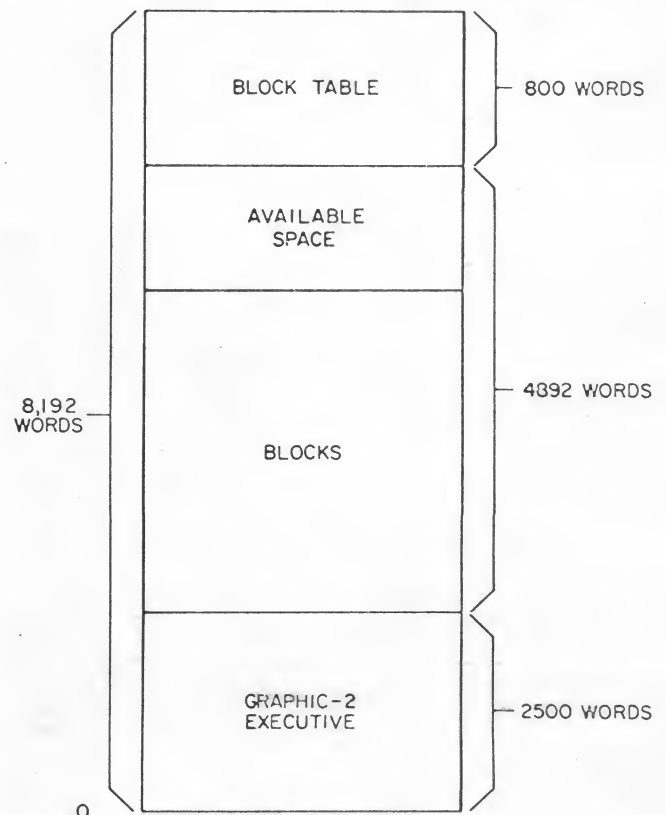


Figure 14—GRAPHIC-2 memory layout

The first time an inter-block reference is executed a search of the block table is made and the ID in the reference is replaced by its corresponding block table entry address (*BTEA*). Succeeding executions of that reference made before either of the blocks is removed from PDP-9 core, may be accomplished without having to search the block table. If the referenced ID is not in the block table then the block is not in core and a request for it is issued to the GE-645.

When a block is removed from core, its entry in the block table is also removed. At this time a search of every block in core is made to find any *BTEA* references to the removed block. These *BTEA* references are then charged to ID's. The hole created by removing a block is added to the available space (see Figure 14) by relocating all the blocks above it. The blocks are always contiguous in core.

In summary, when a block is not in core, all references to it are indicated by use of its ID. When a reference is made to a block not in core, the block is fetched from the GE-645. An entry is made for the block in the block table and reference by ID is changed to a reference by *BTEA*. Other references to this block from any other blocks are *linked* (ID changed to *BTEA*) only when the reference is made. If the block is removed from core, all linked references are unlinked. The block table allows conversion from ID to *BTEA* (as references are linked) and vice-versa (when a block is removed). It also provides the starting core address of the referenced block so that an offset may be added to complete the reference.

Display management

To display the graphic data structure described in this paper, a semi-interpretive display program is used. A single leaf can be displayed by the scope without intervention by the PDP-9, but threading through the nodes and branches requires push down operations and help from the memory management programs. The display manager supplies this structure tracing function. It seeks out every path to every leaf in the specified structure and displays leaves as it encounters them. It uses the memory management program to link all references in the structure (if any are not yet linked) on the first pass through the structure. After the structure is linked, the interpretive and noninterpretive display functions are overlapped, i.e., while the GRAPHIC-2 scope-processor is displaying a leaf the display manager in the PDP-9 finds the next leaf and prepares to display it. Displaying interpretively has several advantages over a linked self-running display list. The relative positioning vectors in each of the branches, for example, are added up along the path to a leaf and the result put out as a single scope positioning command. This technique is

much faster than executing all individual positioning vectors. Also, the pushdown list kept by the display interpreter contains a real time record of the display path. This record is very valuable when a light-pen strike occurs. Interpretive operation also allows the relative positioning vectors in the branches to be stored as full 18-bit Δx and 18-bit Δy values because scope command op-code bits are not necessary. This allows the GRAPHIC-2 data structure to contain an 18-bit by 18-bit picture, of which any 10-bit by 10-bit section can be displayed on the scope face.

VII. GE-645 graphic software system

Routines in the graphic programming system can be divided into two distinct categories: those which are accessible to a user by call or GRIN-2 language statement and those that are invisible to the user but provide necessary services. In the first category are sub-routines for building and editing the user's Graphic Data Structure and for creating and linking nondisplay information to it. In the second category are programs that handle communication between the GE-645 and all graphical devices, and the Translator programs that convert from internal GE-645 picture representation to formats required by specific devices. A dynamic storage allocator is used to allocate and free blocks that are used in the graphic data structure.

The most complicated device to handle from the point of view of the software system is the GRAPHIC-2. A library of GRAPHIC-2 subroutine blocks is maintained on GE-645 disk storage. These are kept in relocatable, linkable format since the GRAPHIC-2 executive system includes a linking loader that runs on the PDP-9. The equivalent routines in the GE-645 version are also kept on a library file. A critical element in the system is the "unique ID maintainer." In Section VI on the GRAPHIC-2 executive system it was pointed out that all program and data blocks in the GRAPHIC-2 have a 17-bit ID number associated with them. A complete dictionary of all blocks with assigned ID's is kept in the GE-645. This dictionary establishes a unique correspondence between blocks in or referred to in the GRAPHIC-2 and the equivalent block in the GE-645. Whenever GRAPHIC-2 needs a block of any kind it asks for it by ID number. The ID table must be used to locate the desired block in the GE-645. The block is then converted to GRAPHIC-2 format (if it is part of the Graphic Data Structure) or retrieved from the GRAPHIC-2 program file (if it is a program block) and sent over the communication link to the remote terminal. The unique ID assigned to a block is a dynamic operation that occurs during program execution and is completely invisible to the programmer.

The "real-time-input" simulator in the GE-645 is invoked wherever one of the real-time GRIN-2 statements is encountered in the GE-645 program. It simply supplies the next argument (or arguments) found on the input queue sent over from GRAPHIC-2. This allows the GE-645 version to undate its version of the Graphic Data Structure to correspond with events taking place at the remote terminal.

SUMMARY

A flexible graphics system that provides users with several different types of service has been described. Several devices of each available type are provided to insure good access to the system for many people. The system is intended for use in an environment where users do most of their own programming. Therefore, software support is provided that makes programming graphical I/O fairly simple.

A wide range of applications is expected. These in-

clude data plotting, circuit mask design and analysis, generating of motion pictures, text editing, and many more.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the contributions of many colleagues in the Information Processing Research and Machine Aids to Development Departments. In particular, we wish to thank A. D. Hause for valuable suggestions during the design of the system and L. Rosler for helping to crystallize many features of the GRIN-2 language.

APPENDIX

The following is a commented example of a graphically interactive program written in GRIN-2. This program allows an operator at a GRAPHIC-2 terminal to draw, move or delete broken line segments on the face of the scope with a light-pen.

1	START	LEAF	DRAWLB
2		TEXT	(D,R,A,W)
3		LEAF	MOVELB
4		TEXT	(M,O,V,E)
5		LEAF	DLTLB
6		TEXT	(D,E,L,E,T,E)
7		NODE	LTBTNS((B1,,DRAWLB,DRAWPG)
			(B2,(0,-30),MOVELB,MOVEPG)
			(B3,(0,-60),DLTLB,DLTPG))
8	BRANCH		LBTRAN,,(950,950),,LTBTNS
9	NODE		DRAWND
10	BRANCH		DRAWBR,,,DRAWND
11	NEWDSP		(LBTRAN,DRAWBR)
12	HOME	WHICH	,TRA
13		GOTO	HOME
14	DRAWPG	WHERE	WHICHX,DRAWI
15	DRAWI	LEAF	DRLEAF
16		BRANCH	DRBRAN,DRAWND,WHEREX,,DRLEAF
17		DRAW	WHEREX
18		GOTO	HOME
19	MOVEPG	WHICH	,SAME
20		TCMOVE	WHICHB
21		WHERE	WHICHX,HOME
22	DLTPG	WHICH	,SAME
23		DETACH	WHICHB
24		GOTO	HOME

Statement 1 generates a leaf in the data structure and a pointer to that leaf is placed in DRAWLB. The displayable text "DRAW" is then placed in the leaf (STATEMENT 2); statements 3,4,5 and 6 generate two more leaves with the text "MOVE" and "DELETE" in them. Statement 7 generates a node LTBTNS and three branches B1, B2, B3 (one connecting each of the three leaves to the node LTBTNS). These

branches are also declared light buttons with the associated programs DRAWPG,MOVEPG and DLTPG. Different delta xy values are given in the three branches so that the three leaves (light-buttons) will be displayed at different locations on the scope. Statements 8, 9 and 10 then generate two more branches and a node. The total structure generated by statements 1 through 8 is then shown in Figure 15a. The node

DRAWND is a place to attach more structure when the DRAWPG light-button program gets control. Statement 11 specifies that the branches LBBRAN and DRAWBR and all the structure below them are to be displayed (Figure 15b). Control then is held in statement 12 asking the question "WHICH function should be performed next?" The TRA argument on the WHICH statement specifies that when a light-button is picked control should be given to its associated program. If a non-light-button were picked (impossible now since there are none), then control would pass to statement 13 which passes it back to 12 (WHICH). This in effect ignores any non-light-buttons and forces the operator to pick a light-button. Pointing to "DRAW" on the scope gives control to DRAWPG, "MOVE" to MOVEPG and "DELETE" to DLTPG. Assuming the operator picked "DRAW" with the light-pen, the

first question asked by the drawing program DRAWPG is "WHERE should the broken line start?" The argument WHICHX in statement 14 is a pair of system locations which contain the scope coordinate of the light-pen strike that answered the last WHICH question. In this case this is the x,y location of where the "DRAW" light-button was touched. This initializes the tracking cross to appear under the light-pen. The operator then moves the tracking cross to the place he wants to start drawing. Pushing a button on the keyboard causes control to pass to DRAW1 as specified by the second argument in statement 14. At this point the scope coordinates of the tracking cross are contained in a pair of system locations WHEREX. DRAW1 then generates a leaf DRLEAF and connects it to the node DRAWND with a branch DRBRAN. The initial position is also inserted in the branch with the WHEREX argument. The DRAW statement (17) then gets control along with the starting location WHEREX (answer from the WHERE question, statement 14). The opera-

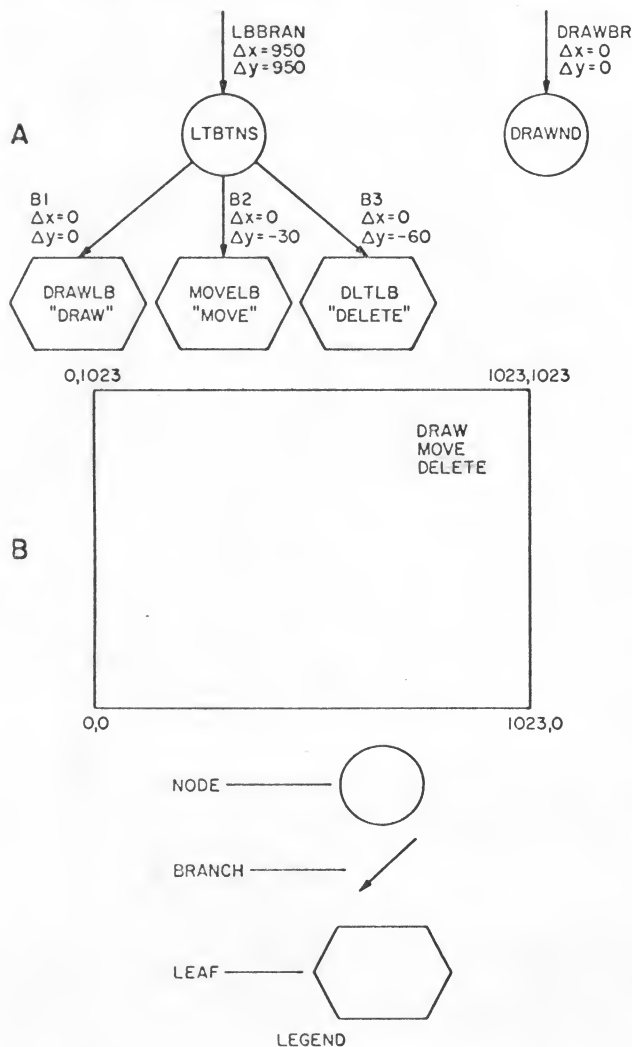


Figure 15—Structure and display for the GRIN-2 example (Appendix)

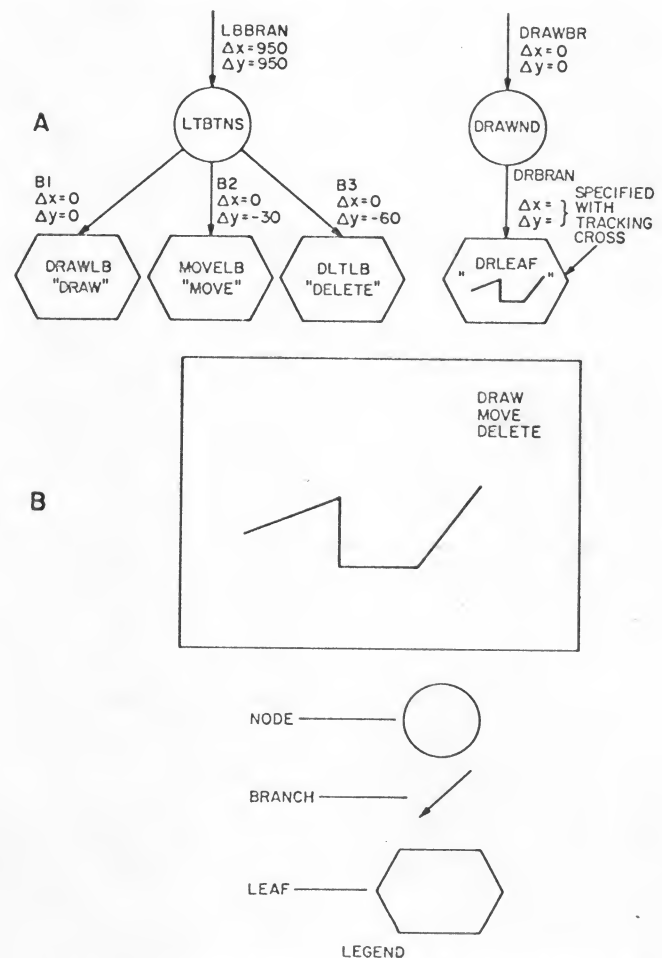


Figure 16—Structure and display for the GRIN-2 example (Appendix)

tor uses two buttons on the keyboard, one to indicate corners and one to indicate the end of the broken line segment. By repeated calls to WHERE and VECTOR, the DRAW statement allows the operator to draw on the scope. When he pushes the end-of-line button on the keyboard, control is passed to the next statement (18) which passes control back to the portion of the program which asks "WHICH function should be performed next?" Figure 16 shows the structure and picture at this point.

If the "MOVE" light button is picked with the light pen, MOVEPG will get control. This program then asks "WHICH branch should be moved?" (statement 19). The "SAME" argument specifies that light-buttons should be treated the same as non-light-buttons. In this case it means the light-buttons, as well as the non-light-buttons, can be moved around the scope. When the operator picks a branch to be moved, control passes to statement 20 which "attaches the object picked (answer left in WHICHB) to the tracking cross. The program then asks "WHERE should the branch be moved?" (statement 21). The picked object then moves with the tracking cross as it moves to follow the light-pen. The argument HOME specifies that control should pass to HOME when a button on the keyboard is pushed, indicating that the moved object is positioned to the satisfaction of the operator. The "DELETE" light button program should now be obvious.

REFERENCES

- 1 I E SUTHERLAND
Sketchpad: A man-machine graphical communication system
Proc. AFIPS Spring Joint Computer Conference, vol. 23, pp. 329-346 1963
- 2 E L JACKS
A laboratory for the study of graphical man-machine communication
Proc. AFIPS Fall Joint Computer Conference, vol. 26, pp. 343-350 1964
- 3 G J CULLER B FRIED
The TRW two-station on-line scientific computer
Computer Augmentation of Human Behavior
- 4 K C KNOWLTON
Computer produced movies
System Analysis by Digital Computer, edited by F. F. Kuo and J. F. Kaiser, John Wiley and Sons, New York, Chapter 11, pp. 375-394 1966
- 5 F H HARLOW J P SHANNON J E WELCH
Liquid waves by computer
Science, vol. 149, no. 3688 p. 1092 September 3, 1965
- 6 H C SO
Analysis and iterative design of networks using on-line simulation
System Analysis by Digital Computer, edited by F. F. Kuo and J. F. Kaiser, John Wiley and Sons, New York, chapter 2, pp. 34-58 1966
- 7 W R SUTHERLAND
The on-line graphical specification of computer procedures
Lincoln Laboratory Technical Report no. 405, Lexington, Massachusetts 1966
- 8 W H NINKE
GRAPHIC-1: A remote graphical display console system
Proc. AFIPS Fall Joint Computer Conference vol. 27, pp. 834-846 1965
- 9 F J CORBATO V A VYSSOTSKY
Introduction and overview of the multics system
Proc. AFIPS Fall Joint Computer Conference, vol 27, pp. 185-196 1965
- 10 E L GLASER
System design of a computer for time-sharing applications
Proc. AFIPS Fall Joint Computer Conference, vol. 27, pp. 197-202 1965
- 11 V A VYSSOTSKY F J CORBATO R M GRAHAM
Structure of the multics supervisor
Proc. AFIPS Fall Joint Computer Conference, vol. 27, pp. 203-212 1965
- 12 R C DALEY P G NEUMANN
A general-purpose file system for secondary storage
Proc. AFIPS Fall Joint Computer Conference, vol. 27, pp. 203-212 1965
- 13 J F OSSANNA L MIKUS S D DUNTEN
Communications and input-output switching in a multiplex computing system
Proc. AFIPS Fall Joint Computer Conference, vol. 27, pp. 213-230 1965
- 14 H S McDONALD W H NINKE D R WELLER
A direct-view CRT console for remote computing
Digest of Technical Papers, 1967 International Solid-State Circuits Conference, vol. 10, pp. 68-69.
- 15 W H NINKE
A satellite display console system for a time-shared computer
In Preparation
- 16 D T ROSS J E RODRIGUEZ
Theoretical foundations for the computer-aided design system
Proc. AFIPS Spring Joint Computer Conference, vol. 23, pp. 305-322 1963
- 17 L G ROBERTS
Graphical communication and control languages
Proc. Information Systems Sciences Second Congress, pp. 211-217 1964